New Paltz
STATE UNIVERSITY OF NEW YORK

Enhanced DDS

Course: Design & Verification of System on chip EGC455-01

Fall 2022

By Christopher Lepore

Table of Contents

Introduction

Direct Digital Synthesis (DDS)

Direct Digital Synthesis (DDS) is a method that produces a wave in digital form. This wave can then be converted to analog through digital-to-analog (DAC). Since the operations are digital, the DDS can quickly switch between output frequencies. In addition, this allows for fine frequency resolution and large range of frequencies. A DDS functions by using a phase accumulator which continuously adds a tuning word value to the count. Then count can be converted to an amplitude using several methods. One of these methods can be a ROM lookup table. This digital amplitude can be converted to analog through a DAC.

FPGA device

A FPGA is an acronym for Field Programmable Gate Array. FPGAs are semiconductor devices within the family of PLDs (programmable logic device). FPGAs are not like ASIC or ASSP which are fixed circuits. A FPGA is based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. They are reusable and can be programmed many times; however, one-time programable (OTP) FPGAs do exist. The FPGA used in this project is the DE10-Lite. This model has a very fast start-up time for FPGAs.

Logic-elements (LE) are the smallest unit of logic in a FPGA. A LE is a digital logic circuit which contains components such as a lookup table (LUT), programable register, and carry and register chains. The carry and register chains can connect carry bits and register outputs to other LEs. When multiple LEs are grouped together a logic-block structure is formed. The logic-block structure is capable of performing larger functions.

Design Tools

The software used to program the FPGA is Quartus Prime Lite Edition and the hardware descriptive language (HDL) used is Verilog. Verilog was used to develop the custom HDL files which became synthesized into hardware blocks. In addition, some Quartus tools used include Single Tap Logic Analyzer and IP (intellectual property) library. The Single Tap Logic Analyzer tool provided a way to verify the FPGA design and functionality. The IP library was used to generate necessary modules quickly.

Design

In this project, an enhanced DDS system was designed. This design is capable of producing a changeable frequency output (range of 0 to 1 MHz) in 1 kHz increments. The variable frequency data is provided through 10 output pins which can be used to interface with a DAC module. Using a 2-bit select input, the 10-bit output for a DAC can be in the form of a triangle, sine, or sawtooth wave. The output frequency is determined using SW9 through SW0 and is updated when KEY0 is pressed. The output frequency is also shown on the 7-segment displays. The most significant bit of the “adjustable accumulator” is used for the square wave output. Additionally, the most significant bit of the “accumulator for 1 Hz” is used for the 1 Hz LED output.

On a reset button (KEY1) press, the output frequency is set to 0 and both accumilators’ counts are set to 0 which will set the square wave output, and the 1 Hz LED to low (while reset is continuously pressed). In addition, while pressed the 7-segment displays will present blank and the PLL clock output will be set low.

The input clock for the schematic is 50 MHz which is stepped down using a PLL to a 10 MHz clock for all internal components. The 10 MHz is also outputted as part of the DAC interface.

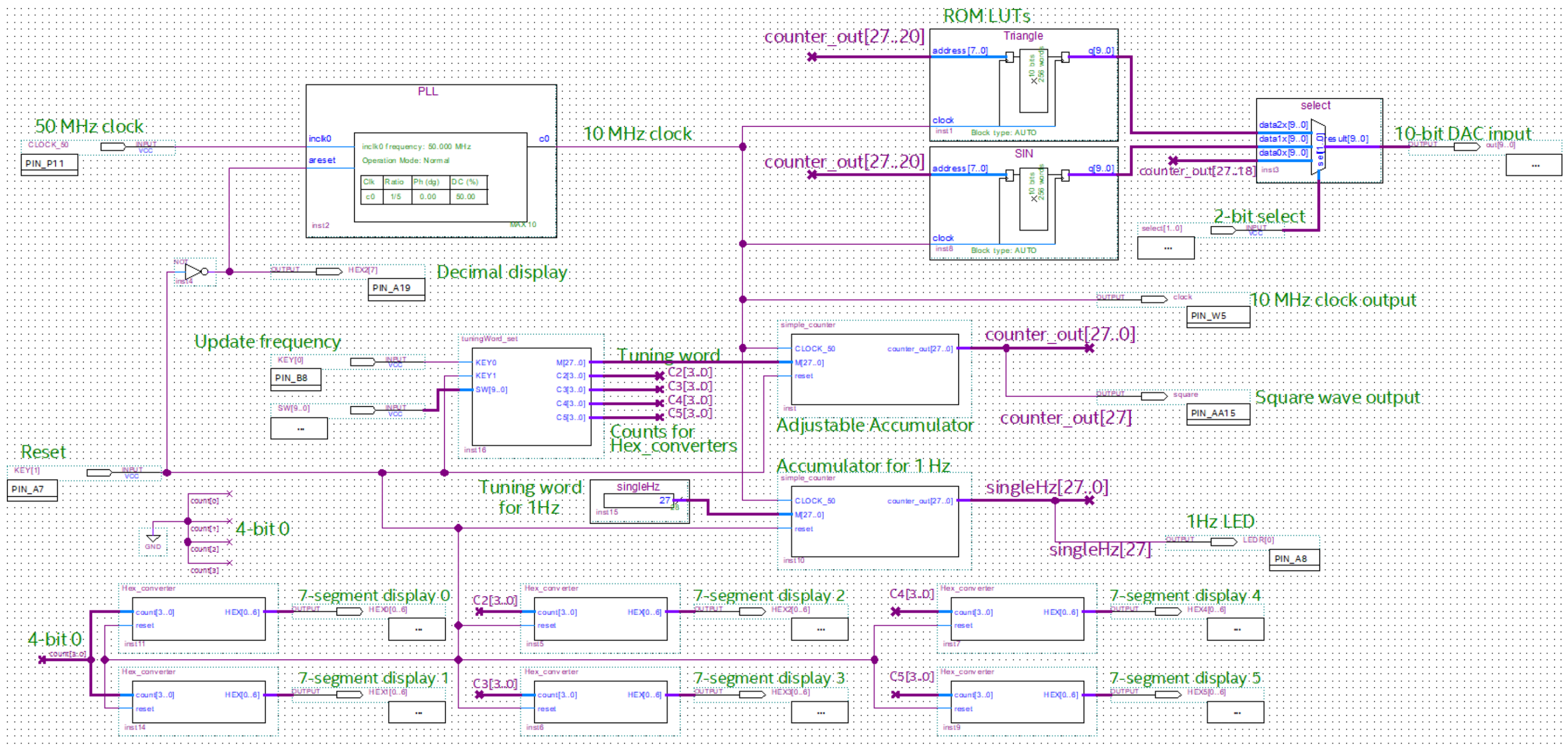


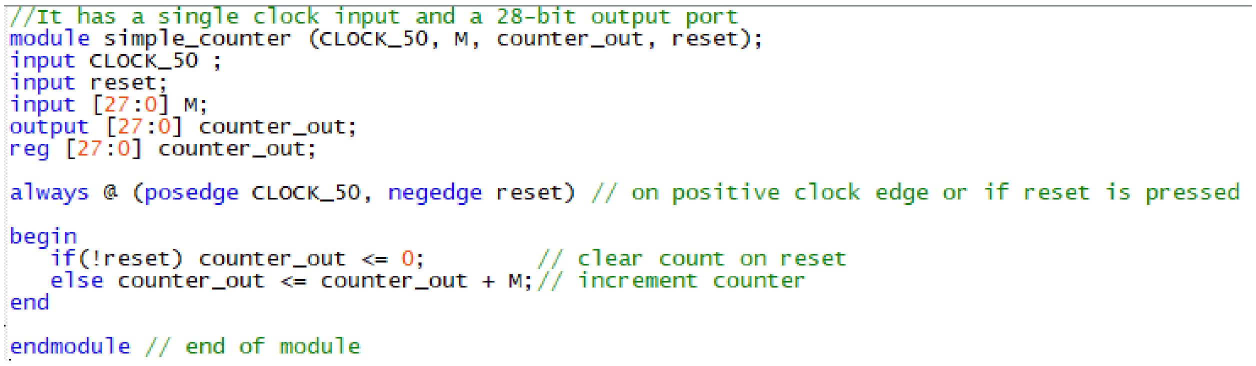
Figure 1. Top-level schematic

Simple_counter (phase accumulator)

The Simple_counter module is a phase accumulator which increments a 28-bit count by the 28-bit tuning word (M), the count is then outputted. This operation will update on the positive edge of the clock input. However, if the reset is pressed then the count will be set to 0. There are 2 instances of this module in the top-level schematic, the “adjustable accumulator” and the “accumulator for 1 Hz”. Each of the accumulators uses the 10 MHz clock.

The “adjustable accumulator” instance generates a 28-bit count_out variable. The variable determines the 10-bit DAC input. The most significant count_out bit is outputted as the square wave output. The tuning word used in this instance comes from the TuningWord_set module.

The “accumulator for 1 Hz” instance generates the 28-bit singleHz variable, the most significant bit is used for the 1 Hz LED output. The tuning word is set to the constant of 27.



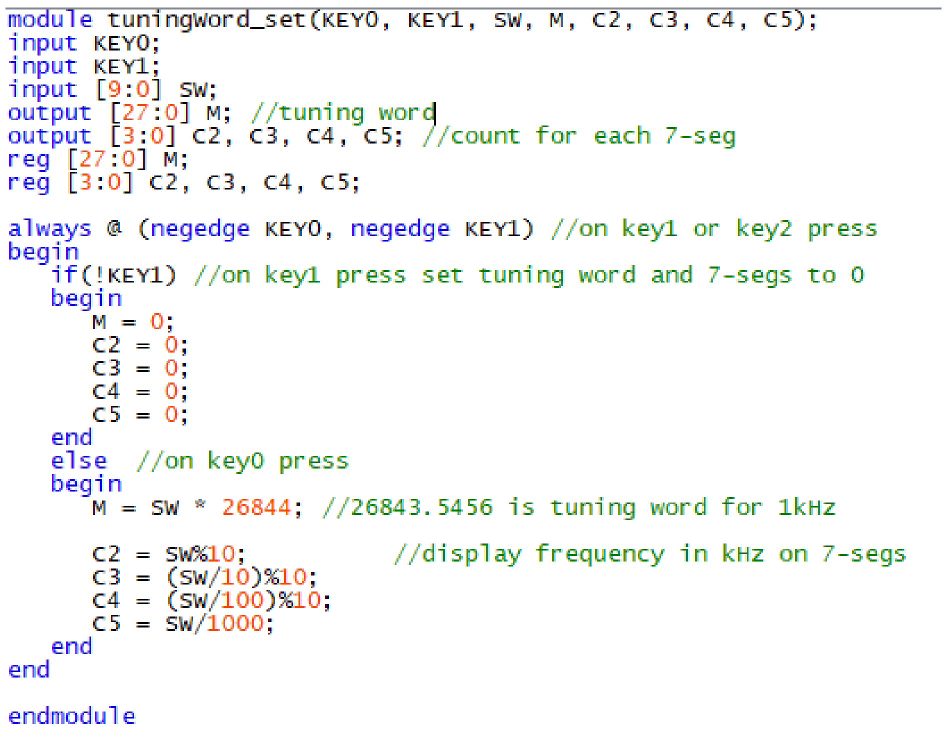
```
//It has a single clock input and a 28-bit output port
module simple_counter (CLOCK_50, M, counter_out, reset);
input CLOCK_50 ;
input reset;
input [27:0] M;
output [27:0] counter_out;
reg [27:0] counter_out;

always @ (posedge CLOCK_50, negedge reset) // on positive clock edge or if reset is pressed

begin
   if(!reset) counter_out <= 0;         // clear count on reset
   else counter_out <= counter_out + M;// increment counter
end

endmodule // end of module
```

Figure 2. Simple_counter HDL file (phase accumulator)

TuningWord_set

This module’s operation is responsible for determining the values for the 4 most significant 7-segment displays. In addition, the module is responsible for the tuning word (M) for the “adjustable accumulator” using switches 0 to 9. The tuning word is calculated by taking the total binary value of the switches (SW9 as most significant) and multiplying it by the tuning word for 1 kHz (~26844). The equation used to determine the tuning word for 1 kHz is: 1 kHz = (M * 10 MHz)/ 2^{28}. The 10 MHz represents the clock input, and the 28 exponent represents the number of bits in the accumulator. The values for each 7-segment display are determined by the total value of the switches. For example, if SW = 2341 then C2 = 1, C3 = 4, C4 = 3, and C5 = 2

(C2-5 are 4-bits); These values are each connected to a Hex_converter for its corresponding 7-segment display. All output registers will only update if KEY0 is pressed, but if KEY1 is pressed then all registers are set to zero.

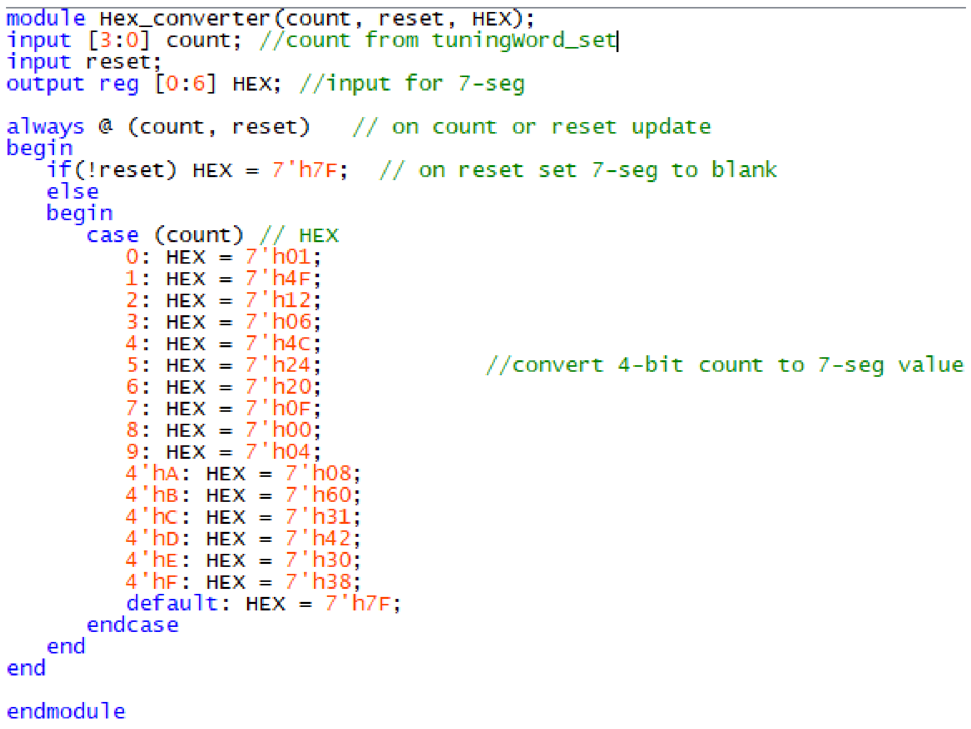


```
module tuningWord_set(KEY0, KEY1, SW, M, C2, C3, C4, C5);
input KEY0;
input KEY1;
input [9:0] SW;
output [27:0] M; //tuning word
output [3:0] C2, C3, C4, C5; //count for each 7-seg
reg [27:0] M;
reg [3:0] C2, C3, C4, C5;

always @ (negedge KEY0, negedge KEY1) //on key1 or key2 press
begin
   if(!KEY1) //on key1 press set tuning word and 7-segs to 0
   begin
      M = 0;
      C2 = 0;
      C3 = 0;
      C4 = 0;
      C5 = 0;
   end
   else  //on key0 press
   begin
      M = SW * 26844; //26843.5456 is tuning word for 1kHz

      C2 = SW%10;          //display frequency in kHz on 7-segs
      C3 = (SW/10)%10;
      C4 = (SW/100)%10;
      C5 = SW/1000;
   end
end

endmodule
```

Figure 3. TuningWord_set HDL file

Hex_converter

The Hex_converter's purpose is to convert a 4-bit number into a 7-bit number for a 7-segment display using a case statement. This module does not control the decimal light of the display. The decimal of display 2 is controlled using the inverse of KEY1 which turns blank on reset. The output will change whenever the count updates, but if the reset is set to low then the display will be set to blank instead of following the case statement.



```verilog
module Hex_converter(count, reset, HEX);
input [3:0] count; //count from tuningWord_set
input reset;
output reg [0:6] HEX; //input for 7-seg

always @ (count, reset)   // on count or reset update
begin
   if(!reset) HEX = 7'h7F;   // on reset set 7-seg to blank
   else
   begin
      case (count) // HEX
         0: HEX = 7'h01;
         1: HEX = 7'h4F;
         2: HEX = 7'h12;
         3: HEX = 7'h06;
         4: HEX = 7'h4C;
         5: HEX = 7'h24;             //convert 4-bit count to 7-seg value
         6: HEX = 7'h20;
         7: HEX = 7'h0F;
         8: HEX = 7'h00;
         9: HEX = 7'h04;
         4'hA: HEX = 7'h08;
         4'hB: HEX = 7'h60;
         4'hC: HEX = 7'h31;
         4'hD: HEX = 7'h42;
         4'hE: HEX = 7'h30;
         4'hF: HEX = 7'h38;
         default: HEX = 7'h7F;
      endcase
   end
end

endmodule
```

Figure 4. Hex_converter HDL file

PLL

A phase-locked loop (PLL) is used to generate, stabilize, filter, modulate, and demodulate frequencies. In this design, the PLL is used to generate a 10 MHz clock from a 50 MHz clock input. The PLL includes a reset which when set high will set the clock to low.

ROM: 1-PORT

In this design, the ROM (read only memory) is used as a lookup table for a triangle and sine wave. The inputs for both instances are the 8 most significant bits of the counter_out variable from the “adjustable accumulator”. Originally, both ROM modules had 1024 words of memory, but this was compressed to 256 words to reduce memory usage. The alteration had no effect on the frequency output because the number of input bits was changed from 10 to 8. The input bits were changed as a result of the reduced words of memory.

LPM_Constant

This module can hold any integer to a determined number of bits. In this instance, the constant integer is 27 in 28-bits. This value is used as the tuning word for the “accumulator for 1 Hz” which outputs the 1 HZ LED.

Multiplexer

The multiplexer (MUX) uses a select input to decide which of its inputs to send to the output. In this instance, the MUX has three 10-bit long inputs and a 2-bit long select input. This MUX determines if the 10-bit output for a DAC is a triangle, sine, or a sawtooth waveform.

Custom HDL modules Vs IP library modules

Using the IP (intellectual property) library can be very beneficial because it contains modules or functions that have already been developed and verified. As a result, it saves time for developers since they don’t have to spend time and effort redesigning functions that have already been proven. Also, the IP library can reduce a design team requirement and make FPGAs available to more groups and applications.

However, the IP library can’t account for every application or requirement. Custom HDL modules allow a developer to create a circuit to their exact requirements and can be as complex as they want. By using both types of modules in conjunction with each other, a developer can efficiently create a design to their specifications.

Verification

All charts were collected using a 2 kHz output frequency with 16k samples.

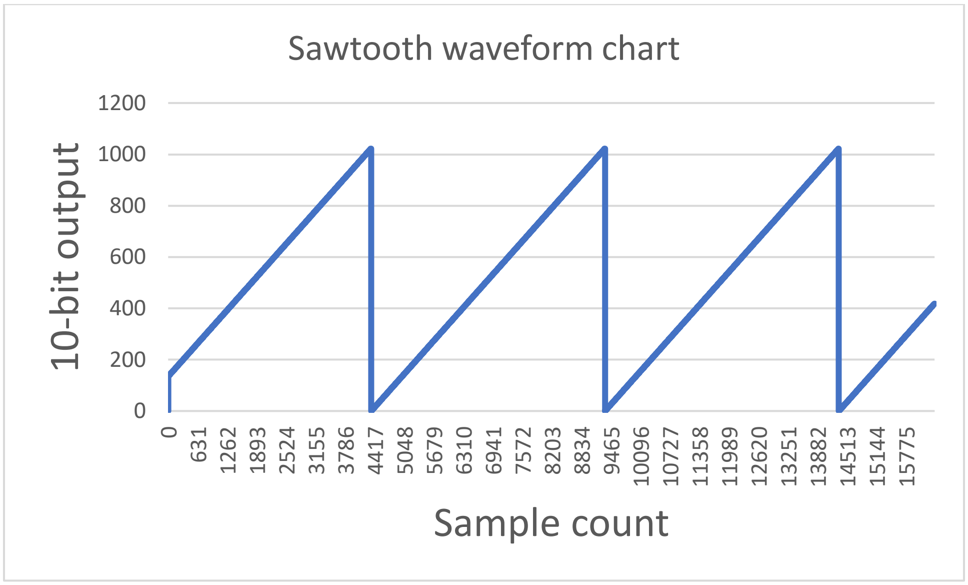


Figure 5. Sawtooth waveform chart

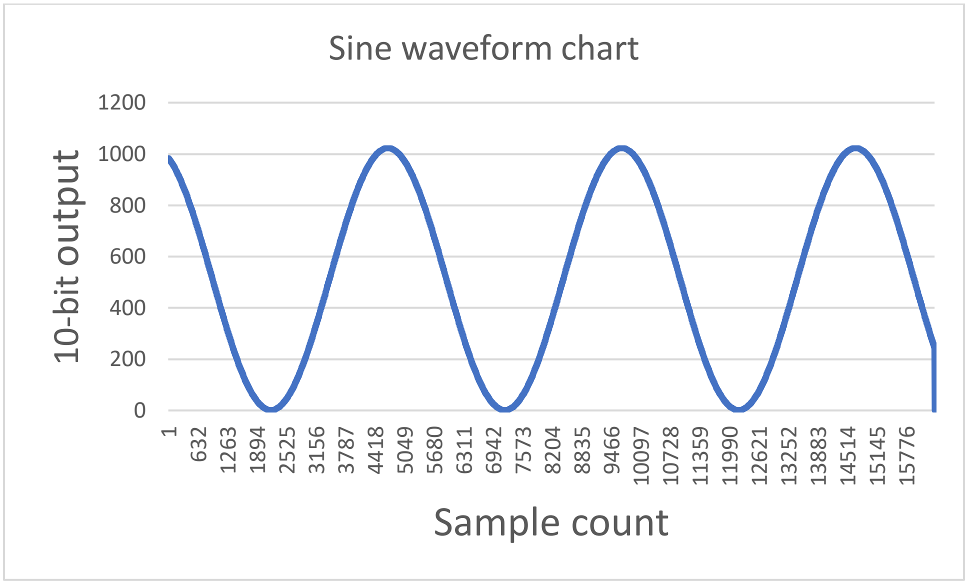


Figure 6. Sine waveform chart

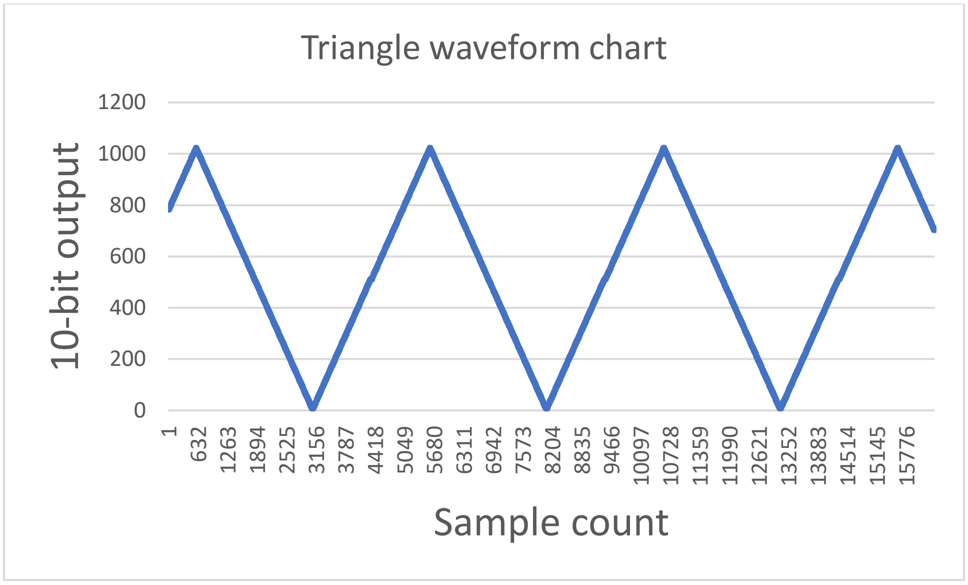


Figure 7. Triangle waveform chart

!!!

Figure 8. 2 kHz square-wave output

Conclusion

In this project, a enhanced DDS system was designed. Through its design and verification, I added skills to my toolbox such as: how to use IP libraries to create modules quickly. In addition, I gained knowledge about the Single Tap Logic Analyzer tool which allowed the verification of DDS 10-bit output. Also, I discovered what a DDS system is and how it functions which will be a great tool for future designs. Some issues encountered in the design process include determining the control for the decimal of 7-segment display 2. This was solved through connecting the decimal output to the inverted KEY1 input which would turn off the decimal light when KEY1 is pressed. Another issue was encountered during the verification process, where the desired clock input was not available. However, this was fixed by extending the window and selecting all pins. Through this project, indispensable knowledge of FPGAs, DDSs, and Quartus was obtained which will be indispensable in future designs.